Big Data and Computing Visions

www.bidacv.com

Big. Data. Comp. Vis. Vol. 4, No. 3 (2024) 219–226.

Paper Type: Original Article

Creation of an Automated Information System Using Python and the Django Framework with a Telegram Bot

Khojiakbar Egamberdiev[1,*], Noila Khidirova[1], Abram Islomov[1]

[1] Department of Computer Systems, University of Economics and Pedagogy, Uzbekistan; ehojiakbar1984@gmail.com; noilaxidirova1989@gmail.com; abramislomov04@gmail.com.

Citation:

Received: 06 April 2024 *Revised: 10 June 2024* *Accepted: 06 July 2024*	Egamberdiev, Kh., Khidirova, N., & Islomov, A. (2024). Creation of an automated information system using python and the django framework with a telegram bot. *Big data and computing visions*, *4*(3), 219-226.

Abstract

This article presents the development of an automated Information System (IS) using Python, the Django framework, and a Telegram bot. The system processes user requests through a chatbot interface, providing data management and retrieval functionalities via a REST API. The integration of Django and Django Rest Framework (DRF) allowed for rapid backend development, while the Telegram bot provided an efficient means of interaction with users. The article describes the system's architecture, development process, and the tools used to implement the solution.

Keywords: Automated information system, Django, Python, Telegram bot, REST API, Data processing.

1 | Introduction

Automated Information Systems (IS) play a crucial role in managing data and processes in modern organizations. Developing such systems using Python and the Django framework allows for rapidly creating powerful web applications and APIs, while integration with Telegram bots provides an intuitive interface for user interaction [1]. This article describes the process of creating an IS that uses Django and the Telegram API to handle user requests through a chatbot.

In the contemporary digital landscape, the demand for efficient and automated information systems has surged across various sectors, including education, healthcare, finance, and customer service. These systems facilitate data management, streamline operations, and enhance user engagement [2]. This paper explores the creation of an automated information system utilizing Python and the Django framework, complemented by a Telegram bot, to provide a seamless user experience and real-time interaction [3].

Django, a high-level Python web framework, is renowned for its robustness, scalability, and built-in security features. It enables rapid development of web applications by promoting the DRY (Don't Repeat Yourself)

Corresponding Author: ehojiakbar1984@gmail.com

https://doi.org/10.22105/bdcv.2024.482114.1210

principle and leveraging reusable components. By harnessing Django's capabilities, developers can create complex web applications that can efficiently manage databases, user authentication, and application logic with minimal effort [4], [5]. The integration of a Telegram bot into this framework offers an innovative approach to user interaction, allowing users to access system functionalities through a familiar messaging platform.

Telegram has emerged as one of the leading messaging applications, boasting over 500 million active users globally. Its bot API provides developers with a powerful tool to create interactive applications that can deliver services directly to users' smartphones. By integrating a Telegram bot into our information system, we aim to enhance accessibility and user engagement, enabling users to interact with the system effortlessly, receive notifications, and access information on-the-go [6].

This paper presents the design and implementation process of an automated information system that encompasses the development of a web application using Django and the creation of a Telegram bot to facilitate user interaction. We will discuss the architecture of the system, the technologies employed, and the challenges encountered during development. Additionally, we will evaluate the performance and user satisfaction of the system through empirical data and user feedback [7], [8].

This study aims to demonstrate the potential of combining Python, Django, and Telegram to create an automated information system that not only meets the functional requirements of modern applications but also enhances user experience through automation and real-time communication [9]. The insights gained from this project could serve as a valuable reference for developers seeking to implement similar solutions in their respective domains.

2 | Methods

For the development of the system, the following tools were chosen.

I. Python: a universal programming language with vast libraries, well-suited for developing web applications, working with databases, and creating APIs [10].

II. Django: a high-level web framework for Python that simplifies backend development. Django offers built-in ORM (object-relational mapping) for database operations, URL management, and REST API support through an additional tool Django Rest Framework (DRF) [11], [12].

III. DRF: an extension of Django for building REST APIs. DRF simplifies data serialization and request handling, making it an excellent tool for API development and bot interaction [13].

IV. SQLite/PostgreSQL: databases used to store information. SQLite is used for development, while PostgreSQL is preferred for production and handling larger data sets [4].

V. Telegram bot: developed using the python-telegram-bot library, the bot serves as the interface for user interaction with the system. The library allows easy integration with the Telegram API and handles commands and messages [6].

VI. Ngrok (optional): used for testing system functionality with webhooks on a local server, providing temporary public access for bot interaction [14].

System architecture

The architecture of the IS consists of the following components.

I. Telegram bot: users interact with the bot through the Telegram app. The bot processes user requests, such as commands or text messages. Polling or webhooks are used to receive new messages from Telegram servers [15].

II. Django application: the Django application serves as the backend, processing requests from the bot and saving and retrieving data from the database. DRF is used to create an REST API with which the bot interacts [16], [17].

III. Database (SQLite/PostgreSQL): stores data about users, their requests, and responses, along with any additional information needed for system operation [18].

IV. REST API: the API, created using DRF, handles requests from the bot, interacts with the database, and sends results back to the bot for user display. For example, the API may receive requests to retrieve user data or store new queries in the database [13].

V. Webhook or Polling: the bot interacts with the system's API through either Polling periodic requests to Telegram servers to check for new messages, or via Webhook, where Telegram automatically sends data to a pre-configured URL upon receiving a new message [19].

Development steps

I. Creating a Django project.

The project is initialized using the following commands.

```
django-admin startproject myproject
cd myproject
python manage.py startapp myapp
```

Database configuration (SQLite for development or PostgreSQL for production) is set in settings.py:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / "db.sqlite3",
    }
}
```

II. Creating data models.

Models define the structure of data to be stored in the database, such as users and their requests:

```
from django.db import models
class User(models.Model):
    telegram_id = models.IntegerField(unique=True)
    name = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
class UserRequest(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    request_text = models.TextField()
    response_text = models.TextField()
    timestamp = models.DateTimeField(auto_now_add=True)
```

III. Database migration.

Applying migrations to create the database tables:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

IV. Setting up the API with DRF.

The API is created to interact with the bot, allowing for data retrieval and submission:

Serializer

```
from rest_framework import serializers
from .models import User
class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['telegram_id', 'name', 'created_at']
```

API view

```
from rest_framework.response import Response
from rest_framework.views import APIView
from .models import User
from .serializers import UserSerializer
class UserDetailView(APIView):
    def get(self, request, telegram_id):
        user = User.objects.get(telegram_id=telegram_id)
        serializer = UserSerializer(user)
        return Response(serializer.data)
```

URL configuration

```
from django.urls import path
from .views import UserDetailView
urlpatterns = [
    path('api/user_data/<int:telegram_id>/', UserDetailView.as_view()),
]
```

V. Creating the Telegram bot:

The bot is registered through BotFather in Telegram, and a token is obtained.

The python-telegram-bot library is installed:

```
pip install python-telegram-bot
```

The code for handling commands is written:

```
from telegram import Update
from telegram.ext import Updater, CommandHandler, CallbackContext
import requests
def start(update: Update, context: CallbackContext) -> None:
```

```
    user_id = update.message.from_user.id
    response = requests.get(f'http://127.0.0.1:8000/api/user_data/{user_id}/')
    if response.status_code == 200:
        data = response.json()
        update.message.reply_text(f"Hello, {data['name']}!")
    else:
        update.message.reply_text("User not found.")
def main():
    updater = Updater("YOUR_TELEGRAM_BOT_TOKEN")
    dispatcher = updater.dispatcher
    dispatcher.add_handler(CommandHandler("start", start))
    updater.start_polling()
    updater.idle()
if __name__ == '__main__':
    main()
```

VI. Testing the system:

Once the Django API and Telegram bot are set up, the system is tested to ensure that the bot interacts with the API correctly, processes requests, and displays data to users [20].

3 | Results

The result is an automated IS capable of receiving user requests via a Telegram bot.

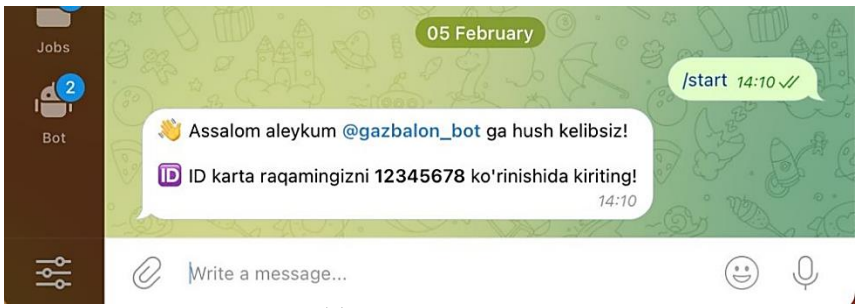

Fig. 1. The view of telegram bot.

And processing them using a web application built on Django.

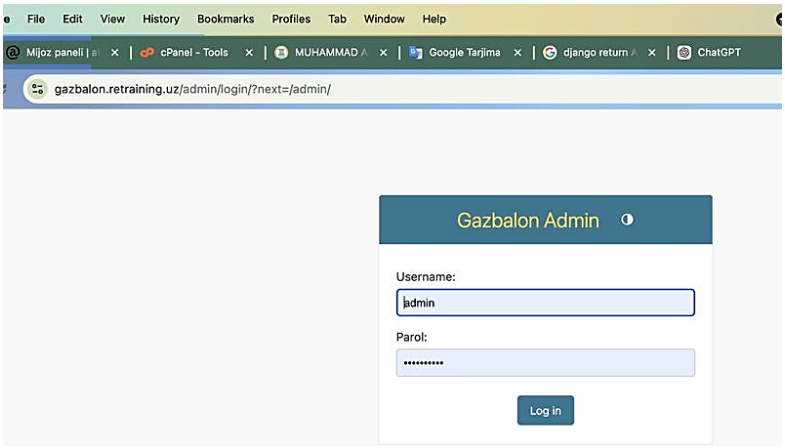

Fig. 2. Login to the admin panel.

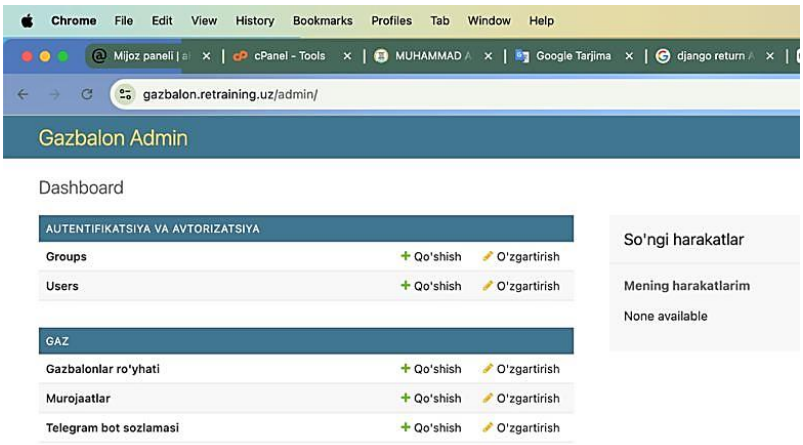

Fig. 3. The view of the admin panel.

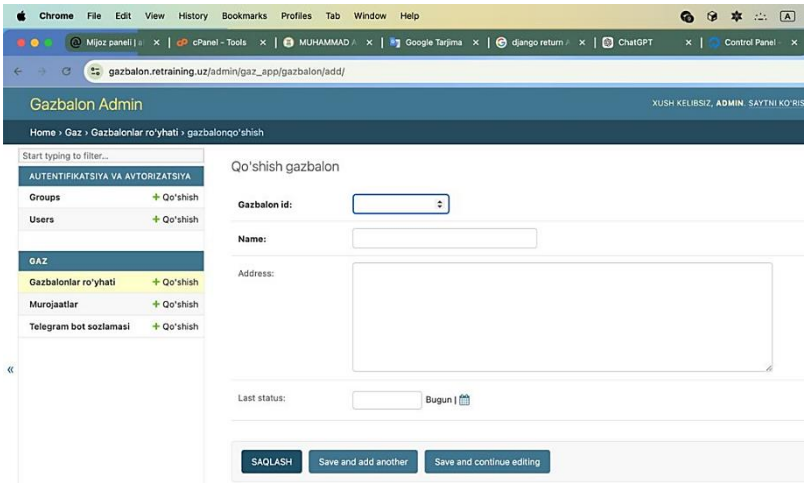

Fig. 4. The window to create a new user with ID.

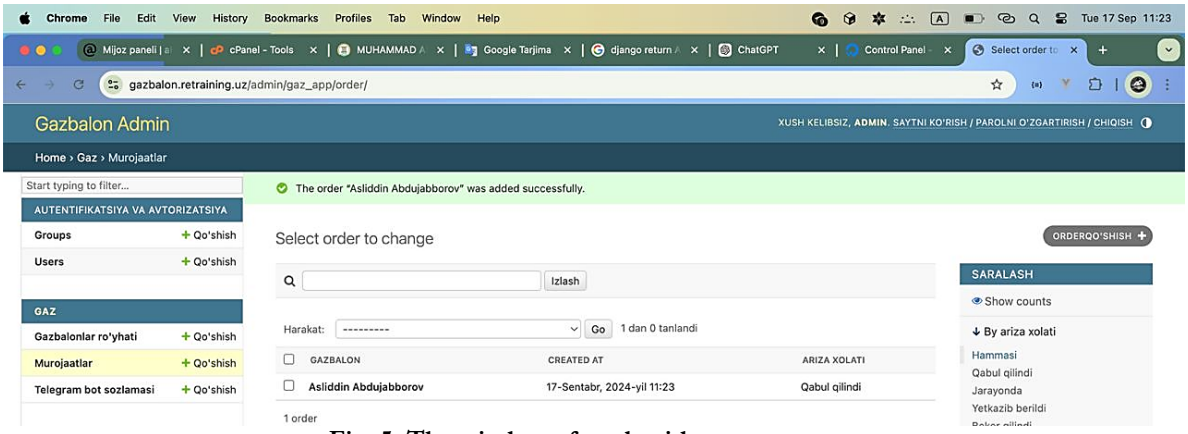

Fig. 5. The window of work with users.

Each user will have their own card with an ID number. The user of this number sends a request to the telegram bot. In turn, the bot sends a request to the employee's personal account.

4 | Discussion

The use of Django and DRF allowed for the rapid development of a flexible API, while the Telegram bot provided a convenient means of user interaction. However, the project encountered challenges such as securing the API and handling large volumes of requests. Future improvements could include scaling the system and adding new features.

5 | Conclusion

Creating an automated IS using Python, Django, and a Telegram bot demonstrated the efficiency of this approach for solving tasks related to data processing and user interaction. This approach can be applied in various industries that require automation of business processes and data management through modern messaging platforms.

Acknowledgments

We would like to express our sincere gratitude to my colleagues who contributed to the development of this project. Special thanks to the Django and Python communities for providing extensive documentation and open-source tools that made this project possible. We would also like to thank the Telegram development team for creating a robust platform that eases the integration of our bot. Lastly, our thanks go to the developers of the DRF for facilitating the development of the API that powers this system.

Author Contributaion

For research articles with multiple authors, provide a short paragraph that identifies each contribution. The following statements should be used: "Conceptualization, Kh.E. and N.Kh.; Methodology, Kh.E.; Software, A.I.; Validation, Kh.E., N.Kh. and A.I.; formal analysis, Kh.E.; investigation, N.Kh.; resources, A.I.; data maintenance, Kh.E.; writing-creating the initial design, A.I.; writing-reviewing and editing, N.Kh.; visualization, Kh.E.; monitoring, N.Kh.; project management, A.I.; funding procurement, Kh.E. All authors have read and agreed to the published version of the manuscript. Authorship must be limited to those who have made a significant contribution to the work reported. All terms were described in the CT file.

Funding

This project is considered a startup and was developed with the help of an investor and the author of the idea. Development progress The automated system is completely finished and at the moment the employees of Gaz Industrials are testing the telegram bot and the system.

Data Availability

Data presented in this study are available upon request from the corresponding author. Data are not publicly available for maintain the privacy of the users of this system. To register each user, you will have to add personal documents. This may be the identity of users will be spread on social networks and the Internet.

Conflicts of Interest

Funders played no role in the design of the study, in the collection, analysis, or interpretation of the data, in the writing of the manuscript, or in the decision to publish the results.

References

[1] Vincent, W. S. (2019). *Django for professionals: production websites with Python & Django*. Independently Publication.

[2] Grinberg, M. (2018). *Flask web development: developing web applications with Python*. O'Reilly Media.

[3] Feldroy, D., & Feldroy, A. (2020). *Two scoops of Django 3. x: best practices for the Django web framework*. Two Scoops Press.

[4] VanderPlas, J. (2016). *Python data science handbook: essential tools for working with data*. " O'Reilly Media, Inc."

[5] Beazley, D., & Jones, B. K. (2013). *Python cookbook: recipes for mastering Python 3*. " O'Reilly Media, Inc."

[6] Documentation, T. A. (2023). *Telegram bot API*. https://core.telegram.org/bots/api

[7] Python Software Foundation. (2023). *Python programming language*. https://www.python.org

[8] Django Rest Framework. (2023). *Django rest framework documentation*. https://www.django-rest-framework.org

[9] Martelli, A., Ravenscroft, A., & Ascher, D. (2005). *Python cookbook*. " O'Reilly Media, Inc."

[10] Sweigart, A. (2019). *Automate the boring stuff with Python: practical programming for total beginners*. No Starch Press.

[11] Sarda, S. (n.d.). *Build REST APIs with Django REST framework and Python: learn basic to advanced Django REST framework by building IMDB API clone (JWT, Token, Throttling, Pagination, Testing)*. https://www.packtpub.com/en-ca/product/build-rest-apis-with-django-rest-framework-and-python-9781801819022?type=video&srsltid=AfmBOoobrmyUjSlpxrXlzInfX6lv4Y8BOvKS5gzDS7h7dnH9u9L-8XBx

[12] Foundation, D. S. (2023). *Django-the web framework for perfectionists with deadlines*. https://www.djangoproject.com/

[13] Alchin, M. (2010). *Pro python*. , Pro Python. Apress.

[14] Ngrok. (2023). *Secure tunnels to localhost*. https://ngrok.com

[15] Brandtzaeg, P. B., & Følstad, A. (2018). Chatbots: changing user needs and motivations. *Interactions*, *25*(5), 38–43. https://doi.org/10.1145/3236669

[16] Baumgartner, P. J., & Malet, Y. (2014). *High performance Django*. Lincoln Loop.

[17] Chun, W. (2001). *Core python programming* (Vol. 1). Prentice Hall Professional.

[18] Reitz, K., & Schlusser, T. (2016). *The Hitchhiker's guide to Python: best practices for development*. " O'Reilly Media, Inc."

[19] McKinney, W. (2022). *Python for data analysis*. " O'Reilly Media, Inc."

[20] Halvorsen, H. P. (2020). *Python Programming*. https://www.saguna.ro/~dorulique/lectzii/2021-2022/11A_Python/Python Programming.pdf